ÉCOLE NATIONALE SUPÉRIEURE
ÉLECTRONIQUE, INFORMATIQUE & RADIOCOMMUNICATIONS
BORDEAUX

2004

Projet de microinformatique

Oscilloscope numérique

BEY Cirine
ZANCAN Adrien

SOMMAIRE

INTRODUCTION 1

MODE D'EMPLOI 1

STRUCTURE DU PROGRAMME 2

I LES DIFFÉRENTS FICHIERS 2
pp.s 2
init.s 2
prog.c 2
def.h 2
peripherique.h 2
prog.h 2
pp.cmd 2

II "PIPELINE" ET BUFFERS 3

II.A) Structure des buffers 3

II.B) Initialisation des buffers 5

III MESURES ET INTERRUPTION TIMER 6

III.A) Trigger 6

III.B) Echelle temporelle 6

IV TRAITEMENT ET PROGRAMME PRINCIPAL 6

V AFFICHAGE ET INTERRUPTION SIOA EN ÉMISSION 7

VI RECEPTION DES CARACTÈRES 7

CONCLUSION 7

INTRODUCTION

Le but du projet est de réaliser un oscilloscope numérique sur processeur 68000, l'écran de l'oscilloscope sera la console de visualisation alphanumérique d'une station SUN.

Pour obtenir une trace stable à l'écran, une fonction "trigger" doit être présente. Aussi, l'utilisateur doit pouvoir modifier les échelles de temps et d'amplitude. Enfin, l'amplitude et la période du signal mesuré doivent être affichées sur l'écran de l'oscilloscope.

MODE D'EMPLOI

L'implantation dans la mémoire du kit se fera grâce à l'instruction :

```
lkit pp.x
```

Le timer doit être validé par l'interrupteur adéquat du kit. Le lancement du programme se fait par :

```
go 1000
```

A l'initialisation, l'oscilloscope est configuré tel ci-dessous :

```
échelle de temps = 1
échelle d'amplitude = 1
trigger actif
trigger en mode automatique
```

Lorsque le trigger est mode automatique, son niveau se place automatiquement entre la valeur maximale et la valeur minimale du signal.

On peut ensuite piloter différentes fonctions de l'oscilloscope par les touches suivantes

Touche appuyée	Action effectuée
'8' ('↑' sur pav. num.)	Augmente l'échelle de temps
'2' ('↓' sur pav. num.)	Diminue l'échelle de temps
'6'('→' sur pav. num.)	Augmente l'échelle d'amplitude
'4'('←' sur pav. num.)	Diminue l'échelle d'amplitude
't'	(Dés)active le trigger
'a'	Active le trigger automatique
'+'	Active le trigger manuel, augmente valeur du trigger
'-'	Active le trigger manuel, augmente valeur du trigger
'q'	Quitte l'oscilloscope

STRUCTURE DU PROGRAMME

I Les différents fichiers

pp.s

Ce fichier assembleur, programme principal, est le point d'entrée du programme. Il contient l'appel au sous-programme d'initialisation ("INIT") et l'appel à la fonction principale ("_main").

init.s

Il contient toutes les initialisations des ports, registres, périphériques et interruptions. Il inclut le fichier "def.h".

prog.c

C'est le fichier qui contient toutes les fonctions réalisées en langage C, toute la programmation de l'oscilloscope y est décrite. Il inclut les fichiers "peripheriques.h" et "prog.h".

def.h

C'est la description de tous les registres du PIT et du SIOA pour leur accès en assembleur.

peripherique.h

Presque comme "def.h", il y est décrit tous les registres du PIT et du SIO mais pour un accès en langage C.

prog.h

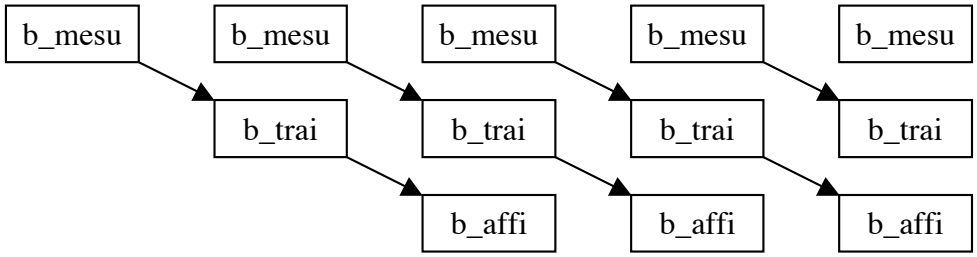
Toutes les structures et les prototypes de fonctions utilisées dans "prog.c" sont déclarées dans ce fichier.

pp.cmd

Ce fichier permet au linker de lier la totalité des fichier aux adresses adéquat. Le lancement du programme se fera à l'adresse $1000.

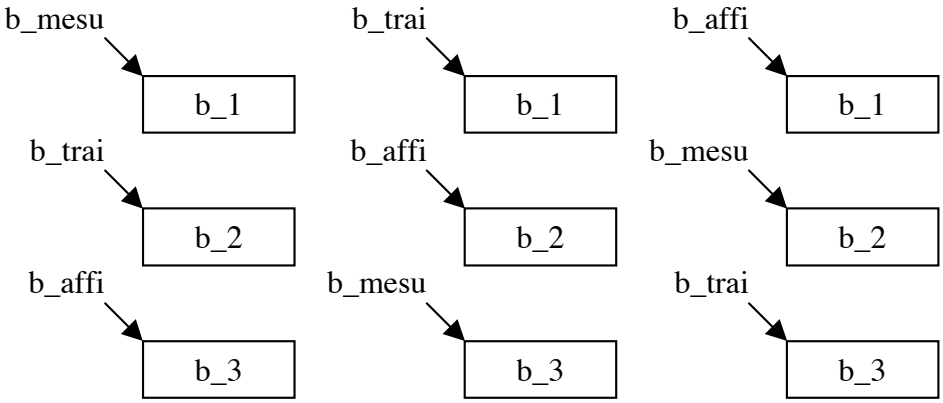
II "Pipeline" et buffers

Le système est dit "pipeliné". Tout le fonctionnement s'articule autour de trois buffers, le buffer de mesures, le buffer de traitement et le buffer d'affichage appelés respectivement *b_mesu*, *b_trai* et *b_affi*. Pendant chaque cycle, 80 valeurs sont échantillonnées et sauvegardées dans le buffer de mesure, pendant que les 80 mesures précédentes sont traitées (conversions ASCII, calcul de l'amplitude et de la période, etc.) et que les 80 mesures précédent celles traitées sont transmises par la liaison série (et affichées à l'écran). A chaque nouveau cycle, il faut recopier le contenu du buffer de traitement dans le buffer d'affichage et celui du buffer de mesure dans le buffer de traitement, ce qui peut-être représenté par le schéma ci-dessous.



Représentation du système "pipeliné"

Cette méthode, appliquée directement, est possible mais fait perdre du temps car il faut à chaque fois parcourir l'ensemble des buffers pour en recopier le contenu. Une manière plus efficace est de déclarer *b_mesu b_trai* et *b_affi* comme des pointeurs vers 3 buffers (appelés b_1, b_2 et b_3) et d'effectuer une permutation circulaire des 3 buffers à chaque cycle, qui peut-être représentée par le schéma ci-dessous.



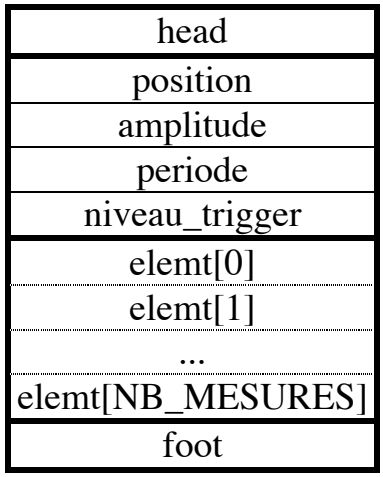
Permutation circulaire des pointeurs

II.A) Structure des buffers

Pour permettre cette permutation circulaire des 3 buffers il faut que ceux-ci soit rigoureusement du même type.

L'envoi du buffer d'affichage sur la liaison série ne doit comporter quasiment aucun traitement, il faut que le code puisse être simplifié au maximum. La meilleure solution est que le buffer d'affichage contienne tous les caractères à envoyer

successivement, ainsi le seul travail de la fonction d'affichage sera de parcourir le buffer d'affichage et de transmettre chacun des caractères qu'il contient. La structure des buffers sera telle représentée ci-dessous.



Structure de chaque buffer

"*head*" correspond aux caractères permettant l'effacement de l'écran.

"*position*" correspond aux caractères servant à positionner le texte affiché.

"*amplitude*", "*periode*" et "*niveau_trigger*" sont de même type et composés de deux chaînes de caractères, la première correspond au texte à afficher, la deuxième correspond à la valeur correspondante à afficher.

"*NB_MESURES*" est une constante définie comme égale à 80, elle correspond aux nombres d'échantillons par cycle mais aussi au nombre de colonnes de notre fenêtre d'affichage.

"*elemt[0* à *NB_MESURES]*" sont les caractères permettant d'afficher les 80 étoiles (images de la mesure) à l'écran. Chaque élément est composé des caractères permettant le placement du caractère '*' à la bonne position sur l'écran (ligne – colonne) ainsi que le caractère '*' même.

"*foot*" correspond au caractère NULL, permettant de détecter facilement la fin du buffer lors du parcours de celui-ci.

Pour une meilleure lisibilité du programme, et donc minimiser le risque d'erreur de programmation, chaque composant d'un buffer à été défini comme "structure" et le buffer lui même est une "structure".

La structure de buffer est ainsi définie :

```
typedef struct
{
  type_buf_head           head;

  type_buf_position_texte position;
  type_buf_texte          amplitude;
  type_buf_texte          periode;
  type_buf_texte          niveau_trigger;

  type_buf_element        elemt [ NB_MESURES ];

  type_buf_foot           foot;
} type_buffer;
```

et la structure "*elemt*" est définie comme suit :

```
typedef struct
{
  volatile unsigned char curseur    [2];
  volatile unsigned char ligne      [2];
  volatile unsigned char point_virgule ;
  unsigned char padding;
  volatile unsigned char colon      [2];
  volatile unsigned char fin_curseur   ;
  volatile unsigned char caract_affiche;
} type_buf_element;
```

Lors de la mise en place de ces structures, il est apparu une erreur que nous n'attendions pas. Lorsque, dans une structure, on déclare un carctère "*char caractere*", suivi de la déclaration d'une chaîne "*char chaine[2]*", les données sont cadrées en mémoire et un "*char*" supplémentaire est introduit automatiquement entre "*caractere*" et "*chaine*". Cette opération est, à priori sans conséquence vu que totalement transparente pour le programmeur mais s'avère désastreuse lorsque l'on transmet le buffer d'affichage caractère par caractère (donc octet par octet) par liaison série, y compris les caractères de cadrage qui ont des valeurs inconnues!

Pour éviter ce problème, nous avons donc ajouté nous-même les caractères "*padding*" aux endroits où ils étaient insérés automatiquement. Ainsi, nous pouvons les initialiser à une valeur connue et non utilisée (nous avons choisi 0xff, cette valeur correspond au caractère 'ÿ' que nous n'utilisons pas). Lors de la transmission du buffer d'affichage, nous n'envoyons donc pas les caractères de valeur 0xff.

II.B) Initialisation des buffers

La fonction :

```
void init1buf ( type_buffer * buf );
```

permet le remplissage du buffer passé en argument au valeurs qui doivent être affichées à l'identique à chaque cycle. Une boucle permet notamment de remplir les champs "*colon[0* à *1]*" à un nombre (codé en ASCII) croissant de 01 à 80, correspondant au balayage de tout l'écran.

III Mesures et interruption Timer

Pour éviter de perdre du temps à effectuer une scrutation mais surtout pour avoir une échelle de temps précise, l'échantillonnage du signal analogique est effectué par interruption Timer. Le microprocesseur effectue ainsi une mesure toutes les 100ns, le redémarrage de la conversion du CAN se fait de suite après avoir récupéré la mesure précédente.

Chaque mesure est sauvegardée dans le buffer de mesure, à la place où se trouvera ensuite la valeur ASCII correspondant à la ligne d'affichage.

```
b_mesu->elemt[ indice_mesu ].ligne[1] = MESURE;
```

Cette place dans le buffer peut paraître étrange, mais nous avons pensé qu'il ne servait à rien de créer un autre champ pour la mesure alors que les valeurs des lignes ne sont pas utilisés avant le traitement.

"*indice_mesu*" correspond à l'indice de parcours de "*b_mesu*".

III.A) Trigger

Après avoir effectué la mesure, si le trigger est activé, le programme entre dans la fonction trigger.

La fonction trigger attend que trois mesures aient été effectuées pour tester si elles passent par le niveau du trigger et que c'est un front montant. Si c'est le cas, les mesures continuent normalement, sinon, la première mesure est rejetée et les deux suivantes sont décalées au début du buffer de mesure.

III.B) Echelle temporelle

L'échelle temporelle est réglée dans l'interruption Timer, un compteur ("*compteur_echelle_temps*") permet de n'effectuer qu'une mesure toute les "*echelle_temps*". Ainsi, si "*echelle_temps = 1*", on aura une mesure par 100ns; si "*echelle_temps = 2*", on aura une mesure par 200ns.

IV Traitement et programme principal

Après les initialisations, le programme principal est une boucle infinie. Lorsque les deux opérations mesures et affichage sont terminés, ils sont redémarrés et la fonction traitement ("*trait1buf()*") est appellée.

Cette dernière consiste tout d'abord à mettre la mesure à l'échelle de l'écran, la convertir en ASCII et la sauvegarder à l'endroit adéquat du buffer.

Par ailleurs, elle détecte les valeurs maximales et minimales ainsi que leurs indices, permettant ainsi de calculer les mesures d'amplitude, de période et le niveau du trigger automatique. Ces valeurs sont ensuite converties en ASCII.

V Affichage et interruption SIOA en émission

Pour ne pas perdre de temps à la scrutation de flags, la transmission série est faite en interruption. Le buffer d'affichage est parcourue dans son ensemble et tous les caractères sauf les caractères "*padding*" (0xff) sont envoyés un par un à chaque interruption. Pour simplifier la fonction, le buffer est "casté" en "*unsigned char **" et parcouru comme un simple tableau de caractères.

VI Reception des caractères

Toujours pour avoir un système optimal, la réception des caractères entrés au clavier de la station SUN génère une interruption et donc un traitement immédiat.

Une simple structure de type "*Switch...case*" permet d'effectuer telle ou telle opération suivant la touche appuyée.

La liste des actions associées aux touches est signalée dans la partie "Mode d'emploi" (page 1).

CONCLUSION

Ce projet nous a donné l'opportunité de réellement mettre en pratique la microinformatique apprise en cours et en TP. Bien que l'approche d'un oscilloscope sur console alphanumérique soit plutôt étonnante, nous avons trouvé le sujet intéressant, notamment en nous apprenant des techniques qui sont très courantes, comme le pipeline et le fonctionnement en interruptions de tous les périphériques.

Cependant, même si nous avons trouvé le sujet trop long par rapport au temps imparti, nous savons pertinemment que le temps passé à ce projet ne peut-être que bénéfique, aussi bien pour notre scolarité que pour notre avenir professionnel.

ANNEXES

pp.s	Annexe 1
pit.h	Annexe 2
init.s	Annexe 3
peripheriques.h	Annexe 4
prog.h	Annexe 5
prog.c	Annexe 6
pp.map	Annexe 7